Freescale Semiconductor, Inc.

OSEKturbo OS/MPC5xx v.2.2.1

User's Manual

Because of last-minute software changes, some information in this manual may be inaccurate. Please read the readme.txt file for the latest information.

Revised: June 2003

For More Information: www.freescale.com

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and B are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Employment Opportunity/Affirmative Action Employer.

Legal Notices

The information in this document has been carefully checked and is believed to be entirely reliable, however, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Motorola does not assume liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.

The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

No part of this publication may be reproduced or transmitted in any form or by any means - graphic, electronic, electrical, mechanical, chemical, including photocopying, recording in any medium, taping, by any computer or information storage retrieval systems, etc., without prior permissions in writing from Motorola Inc.

Permission is granted to reproduce and transmit the Problem Report Form, the Customer Satisfaction Survey, and the Registration Form to Motorola.

Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, Motorola assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. Motorola further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. Motorola disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, including implied warranties of merchantability or fitness for a particular purpose.

Trademarks

OSEK/VDX is a registered trademark of Siemens AG.

Diab, D-CC, and SingleStep are trademarks of Wind River Systems, Inc.

Metrowerks, the Metrowerks logo and CodeWarrior are registered trademarks of Metrowerks Inc.

Microsoft, MS-DOS and Windows are trademarks of Microsoft.

Contents

1

Introduction

This User's Manual describes how to install OSEKturbo OS/MPC5xx, and to build sample and user's applications. Information about OSEK services and OIL parameters is provided.

"Installation" chapter describes how to install OSEKturbo OS/MPC5xx.

"Sample Application" chapter provides the user with definition of the sample application and instructions how to build the sample application.

"Tutorial" chapter contains description how to create a new simple application.

"Using an Unsupported Target Derivatives" chapter contains recommendations about OSEK OS adaptation to other derivatives.

"Quick Reference" appendix lists OSEK OS run-time services with entry and exit conditions as well as OIL object parameters with their possible values and short descriptions.

This chapter consists of the following sections:

OSEK OS Overview

OSEK Operating System is a real-time operating system which conforms to the OSEK OS v.2.2 specification.

The OSEK OS meets the following requirements:

- OS is fully configured and statically scaled;
- OS performance parameters are well known;
- The most part of the OS is written in strict correspondence with ANSI C standard, the OS and the application on its basis can be easily ported from one platform to another.

A wide range of scalability, a set of system services, various scheduling mechanisms, and convenient configuration features make the OSEK Operating System feasible for a broad spectrum of applications and hardware platforms.

The OSEK OS provides a pool of different services and processing mechanisms for task management and synchronization, data exchange, resource management, and interrupt handling. The following features are granted to the user:

The OSEK OS is built according to the user's configuration instructions while the system is generated. System and application parameters are configured statically. Therefore, a special tool called the System Generator is used for this purpose. Special statements are designed to tune any parameter. The user must only edit the definition file, run the System Generator and then assemble resulting files and application files. Thus, the user can adapt the Operating System to the control task and the target hardware. The OS cannot be modified later at execution time.

Technical Support Information

To order Motorola/Metrowerks products or literature, consult your local sales representative.

For technical support please use:

US
Tel: +1 512 997 4700
Fax: +1 512 997 4901
Email: support@metrowerks.com

Europe
Tel: +41 61 69 07 505
Fax: +41 61 69 07 501
Email: support_europe@metrowerks.com

Web: http://www.metrowerks.com/MW/Support

Download updates at
http://www.metrowerks.com/MW/Support/Download

2

Installation

The chapter contains information about the environment required to install the OSEKturbo OS and describes installation/uninstallation.

This chapter consists of the following sections:

Preface

This version of the OSEKturbo OS is to be used on an IBM PC 486 (and higher) compatible. The PC must work under MS Windows 2000/98 during the OSEK installation.

The full size of the OSEKturbo OS file set is 9 MB. To install the product, there may be required up to 18 MB of hard disk space depending on the used file system. At least 2 MB of disk space is required to run SETUP.EXE. About 25 MB of disk space is required for the temporary files during installation.

The OSEKturbo OS installation is protected with FLEXcrypt for Windows 2000/98. The OSEKturbo OS System Generator utility is protected with FLEXlm. To get the installation decryption key and the license file for OSEK OS SysGen utility and TargetDLL, please register. Metrowerks provides a registration tool for easy registration (MWRegister).

To register, start MWRegister.exe in the subfolder 'MWRegister_OSEK_MPC' or in the folder you have choosen during a CD installation, fill out the form and press 'Register'. Select a method for registration and press 'OK'.
Please provide your registration number/license number (if available) in the form to accelerate the registration process.

NOTE NOTE: It is important to run MWRegister on the machine where OSEKturbo shall be installed.

For more help on MWRegister, please read MWRegister_ReadMe.txt in the same directory.

It is strongly recommended to close all other programs and login as Administrator before installation. It helps to avoid an access error during shared files and system icons installation and updating the Windows Registry.

It is not recommended to install the OSEKturbo OS into the directory with spaces (like "Program Files"). If the OSEKturbo OS is installed into a directory with spaces, then it is not possible to use makefiles, `msmake.bat` and `gnumake.bat` files located in `SAMPLE` subdirectory.

To use the OSEKturbo OS after installation the Cross Compiler should be installed on your computer. You must call the DOS prompt under Windows 2000/98 to run the Microsoft nmake utility or GNU make utility. All supplied makefiles are developed for the Microsoft C++ nmake and GNU make (from Cygwin package v.1.3.9) utilities.

OSEKturbo OS Installation

To setup the OSEKturbo OS on your hard drive:

1. Insert the installation CD.
2. Run `SETUP.EXE`.
3. Follow prompts and instructions of the installation program.
4. Select directories.

   Target Directory is a directory for OSEK source files, personality files and platform specific SysGen files. It is `c:\metrowerks\osek\ostmpc` by default.

   Shared Components is a directory where System Generator common files are placed. If you have installed any OSEK OS v.2.1 or Builder v.2.2 and higher before the current installation, the setup program proposes to select the existing System Generator path for the SysGen root directory. It is strongly recommended not to

change this path and update the existing SysGen. If the System Generator has not been installed before, you can select any path for the SysGen root directory (`c:\metrowerks\osek` by default).

5. Select components which you want to install. You can choose Custom installation and select OSEK OS components which are to be installed in the Custom Installation dialog box. By default all components are selected.
6. After installation verify the consistency of the package by means of comparing the real set of files and directories with the list in the `filelist.txt` file.

After installation the hard drive should contain the OSEKturbo OS root directory `$OSEKDIR` which will contain a set of files in the following subdirectories:

- `BENCHMARK` - OSEKturbo OS benchmarks for performance and memory measurements
- `BIN` - Platform specific part of the System Generation
- `INC` - Operating System header files
- `MAN` - User's Documentation
- `PF` - Personality files
- `SAMPLE` - OSEKturbo OS Sample application
- `SRC` - Operating System source files

The `$OSEKDIR` directory contains the `filelist.txt` file with a complete list of files included in this release and the `readme.txt` file, which provides additional information for the user.

After installation the hard drive should contain the root directory of the System Generator utility which will contain the following subdirectories with the System Generator and Configuration Tools files:

- `$OSEKSHARED/BIN` - SysGen and Configuration Tools Files
- `$OSEKSHARED/TEMPLATES` - OSEK Builder templates

The following common shared files can be updated during installation:

- `$WINSYSTEM\MFC42.dll`
- `$WINSYSTEM\MSVCP60.dll`
- `$WINSYSTEM\MSVCRT.dll`

These files are redistributed according to the separate License Agreement included in the Visual C++ version 6.0 product

NOTE `$OSEKDIR`, `$OSEKSHARED` and `$WINSYSTEM` are placeholders for the OSEKturbo OS root directory, `Shared Components` root directory and Windows system directory names respectively. They are used in this document as references to the corresponding directories.

Silent Installation mode

In the silent installation mode there is no need for a user to monitor the setup and provide input via dialog boxes. To use the installation in silent mode you should create response file first. To do so you should perform the following steps:

1. Run the SETUP.EXE program from the OSEKturbo OS/MPC5xx installation CD with the follwing command-line options:

   setup -r -f1<path\ResponseFile>
   where the -r option causes Setup.exe automatically to generate a silent setup file (.iss file), which is a record of the setup input; -f1 specifies location and name of the response file (.iss file)

2. Follow the prompts and instructions of the installation program to create response file to repeat actions

To perform installation in silent mode you should run the SETUP.EXE program from the OSEKturbo OS/MPC5xx installation CD with the follwing command-line options:

setup -s -f1<path\ResponseFile> [-f2<path\LogFile>]
where the -r option causes SETUP.EXE to execute in a silent mode; -f1 specifies location and name of the response file (.iss file); -f2 specifies a location and name of the log file. The result of installation indicated in the log file in the [ResponseResult] section after the ResultCode keyname, the ResultCode=0 corresponds to a successful installation.

NOTE Please note that installation in silent mode shall be repeated only in the same condition as the response file was created - if for example you have OSEKturbo OS/MPC5xx installed on your PC and then run installation in silent mode which was created whithout installed OS then the installation will fail as there is no input saved in response file for additional dialog boxes.

License File

If the OSEK OS package(s) has not been installed on your computer, then the received license file should be stored on your hard disk as "`C:\flexlm\license.dat`". If the OSEK OS has been installed on your computer before the current OSEKturbo OS installation, the license file has already existed on the system for the OSEK OS packages used. In this case copy strings with the current OSEKturbo OS features licensed from the received license file into the existing one – simply add the contents of the received file to the existing license file.

If you need to have the license file in another location, use the `LM_LICENSE_FILE` environment variable to define another license file location.

Under Windows 2000/98 it is also possible to use the License File Manager to define non-standard license file location (the License File Manager is automatically installed on your PC by the OSEKturbo OS installation procedure). To do this move the license file into a desired location and run the OSEKturbo OS SysGen utility. The License File Manager dialog will appear providing you with a possibility to browse the license file.

OSEKturbo OS Uninstallation

To uninstall the OSEKturbo OS:

- Use the '`UnInstall` OSEKturbo OS/MPC5xx v.2.2.1 `Build <build number>`' item of the *Add/Remove Programs* module of the Windows Control Panel or the corresponding icon in the OSEKturbo OS/MPC5xx program folder.

- Delete the OSEKturbo OS root directory and all its subdirectories to delete data created during the OSEKturbo OS usage.

3

Sample Application

The chapter presents the sample application and describes how to build the sample application.

This chapter consists of the following sections:

Source Files

The Sample application consists of the following source files which are placed in subdirectories of the `sample\standard` directory:

- `common` - contains derivative independent part of the sample configuration file and the source code:
  - `samplets.c` – the application code (`TASKSND1`, `TASKSND2` and `TASKCNT`).
  - `samplerv.c` – the application code (`TASKRCV1`, `TASKRCV2` and `TASKSTOP`).
  - `sample.h` – header file for the application code.
  - `main.oil` – OSEK Implementation Language file, platform independent part.
- derivative dependent parts of the sample are located in corresponding subdirectories:
  - `cfg555cw.oil` – OSEK Implementation Language file for MPC555 platform and CodeWarrior compiler.
  - `cfg555db.oil` – OSEK Implementation Language file for MPC555 platform and Diab Data compiler.
  - `cfg563cw.oil` – OSEK Implementation Language file for MPC563 platform and CodeWarrior compiler.
  - `cfg563db.oil` – OSEK Implementation Language file for MPC563 platform and Diab Data compiler.

– `cfg565cw.oil` – OSEK Implementation Language file for MPC565 platform and CodeWarrior compiler.

– `cfg565db.oil` – OSEK Implementation Language file for MPC565 platform and Diab Data compiler.

Each OIL file accompanied by the couple of the OSEK Builder configuration files which have the same name and `.app` and `.pws` extensions. These files provide the user with possibility to configure and build the OS with OSEK Builder.

– `msmake.bat` - command file for compiling sample using Microsoft nmake utility.

– `gnumake.bat` - command file for compiling sample using GNU make utility.

The directory structure of the Sample application is described in the `readme.txt` file located in the `sample\standard` directory.

Building Sample

Take the following steps to build the sample application:

1. Open the Windows command prompt window.
2. Change the current directory to `sample\standard\<derivative>` directory which contains sample source files. Hereafter the `<derivative>` term shall be used for meaning the name of the subdirectory which keeps the target specific files. For example, a `mpc555` subdirectory.
3. If you use the Microsoft nmake utility, execute the following command:

```
msmake.bat <compiler>
```

where `<compiler>` is a specific compiler name and can be set to *codewarrior* for CodeWarrior compiler or *diab* for *Diab Data* compiler.

If you use GNU make utility, execute the following command:

```
gnumake.bat <compiler>
```

NOTE If some of compiler, OSEK OS or System Generator files are not found during building, check accuracy of the paths defined in the `sample\standard\common\environment.bat` file.

4. After completion of the building the following subdirectories and files are created in the `sample` directory:

- `gen` subdirectory contains `cfg<target>.c` files, `cfg<target>.h` and `osprop.h` files generated by SysGen, where `<target>` is defined by the derivative and the used compiler like in name of the corresponding OIL file.
- `obj` subdirectory contains object files.
- `bin` subdirectory contains the executable file, linker map and ORTI file.
- To execute the sample application load the executable file placed in the bin subdirectory to the evaluation board using the debugger.
- To clean all files generated during the sample building, execute one of the following commands:

```
msmake clean
gnumake clean
```

4

Tutorial

The chapter describes how to create a new simple application.

This chapter consists of the following sections:

Creating New Application

This example has two tasks which activate each other cyclically. Each of the tasks is placed in a separate source file.

Configuration File

A very simple OIL file will be used in an application. However, it performs configuring of a small application. One application mode will be defined. As much as possible attributes will be omitted (default values will be used by the System Generator).

To create an application source code, take the following steps:

1. Create a new directory, for example `c:\userapp`.
2. Create an OSEK OS configuration file `appcfg.oil` in the directory `c:\userapp`.
3. Add OIL version and OIL implementation include file to the `appcfg.oil` file. The implementation supplied with the OSEK OS is used for this sample. Please, find the implementation file in the `$OSEKDIR\BIN` directory:

```
OIL_VERSION = "2.3";
#include "ost22mpc.oil"
```

4. Add a CPU section to the OIL file. Fill the CPU section with two mandatory objects: APPMODE and OS. Add two TASK objects into the CPU section. Give them names as follows:

```
CPU cpu1 {
     APPMODE Mode {};
     OS os1 {};
     TASK TASKA {};
     TASK TASKB {};
};
```

5. Add eight mandatory attributes to the *OS* section. They define that the application will work in EXTENDED status and no hooks are used:

```
OS os1 {
  STATUS = EXTENDED;
  STARTUPHOOK = FALSE;
  SHUTDOWNHOOK = FALSE;
  PRETASKHOOK = FALSE;
  POSTTASKHOOK = FALSE;
  ERRORHOOK = FALSE;
  USEGETSERVICEID = FALSE;
  USEPARAMETERACCESS = FALSE;
};
```

6. To define the target derivative add the following attributes into the *OS* section:

```
TargetMCU = MPC55500 {
};
```

   TargetMCU attribute defines type of *CPU*.

   Two tasks are to be defined in the *OIL* file. They are full-preemptable Basic tasks. TASKA priority is higher than TASKB priority. TASKA is started automatically by *OS*.

7. To configure tasks add the following attributes into the TASKA and TASKB objects:

```
TASK TASKA {
  PRIORITY = 2;
  SCHEDULE = FULL;
  AUTOSTART = TRUE;
  ACTIVATION = 1;
};

TASK TASKB {
  PRIORITY = 1;
  SCHEDULE = FULL;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
};
```

The *APPMODE* object does not require any attributes.

There are no any resources, ISRs, events and timers in this small application. *BCC1* class is selected automatically by SysGen.

You can find below complete listing of the `appcfg.oil` file:

```
OIL_VERSION = "2.3";
#include "ost22mpc.oil"
CPU cpu1 {
  APPMODE Mode {};
  OS os1 {
    STATUS = EXTENDED;
    TargetMCU = MPC555 {
    };
    STARTUPHOOK = FALSE;
    SHUTDOWNHOOK = FALSE;
    PRETASKHOOK = FALSE;
    POSTTASKHOOK = FALSE;
    ERRORHOOK = FALSE;
    USEGETSERVICEID = FALSE;
    USEPARAMETERACCESS = FALSE;
  };
  TASK TASKA {
    PRIORITY = 2;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
```

```
  };
  TASK TASKB {
    PRIORITY = 1;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
  };
};
```

Source Code

Two source files will be used in the application. Each of them contains one task. Take the following steps to create a source code:

1. Create a file `app1.c` in the directory and add the following code to the file:

```
#include "osprop.h"     /* OS Properties file */
#include <osapi.h>      /* OSEK API declarations */
#include "app.h"        /* application header */
#include <appcfg.h>     /* definitions for system objects */

int main( void )        /* entry point of the application */
{
  StartOS( Mode );      /* jump to OSEK startup */
}
TASK( TASKA )           /* task A */
{
  ActivateTask( TASKB );       /* Activate task TASKB */
    /* TASKB priority is lower than TASKA priority */
    /* Therefore TASKB transfer to ready state by */
    /* ActivateTASK service */
  TerminateTask( );            /* TASKA Terminate itself */
    /* TASKB will be transferred to the running */
    /* state after terminating TASKA */
}
```

This file contains the main function which starts the OS. The TASKA code is placed in this file also. TASKA activates TASKB whose priority is lower and then terminates itself.

2. Create a file `app2.c` in the same directory `c:\userapp` and add the following code to the file:

```
#include "osprop.h"     /* OS Properties file */
#include <osapi.h>      /* OSEK API declarations */
#include "app.h"        /* application header */
#include <appcfg.h>     /* definitions for system objects */
TASK( TASKB )           /* task B */
{
  ChainTask( TASKA );   /* Chain to TASKA */
    /* TASKB is terminated by this service call */
    /* TASKA is activated as a chain task */
    /* TASKA will be transferred to the running */
    /* state after TASKB termination */
}
```

This file contains a TASKB function. This task only chains TASKA. It does not do anything else.

3. Create a header file `app.h` in the same directory. This file is required for message types and user types declarations. Add the following code to the file:

```
#ifndef APP_H
#define APP_H
#endif /* APP_H */
```

4. Copy the file `vector.c` from `$OSEKDIR\hwspec` directory to `c:\userapp` directory. This source file contains the start up code and interrupt vector table definition.

NOTE You can modify the vector.c file in the application directory or create your own application vector table to fit it to a specific application (see "Vector Table"). DO NOT change the vector.c file in the `$OSEKDIR\hwspec` directory. This file is used as a template and for a sample application building.

MakeFile

The Makefile from a sample application included in the OSEK OS package can be used for compiling the example. The Makefile for the Microsoft NMAKE utility is used in this example. If you want to use the makefile for the GNU MAKE utility, follow the instructions but take files of the GNUMAK directory instead of files of the MSMAK one.

Before a makefile construction you have to select a compiler and a platform. The choice defines the template for a new makefile. You can use the following templates from `$OSEKDIR\sample\standard\<derivative>\msmak` directory:

- `cw555.mak` - CodeWarrior compiler and MPC555 CPU
- `db555.mak` - Diab Data compiler and MPC555 CPU

The Diab Data compiler and MPC555 platform were selected for the example application.

To tune the makefile for our new application perform the following actions:

1. Copy file `db555.mak` from `$OSEKDIR\sample\standard\mpc555\msmak` to `c:\userapp` directory.
2. Rename `c:\userapp\db555.mak` file to `c:\userapp\makefile`.
3. Open the file `makefile` in any text editor.
4. Find the fragment beginning with the “Application dependent names” comment.
5. Change the application directory name in the following line:

```
appdir = ..\common
```

   to:

```
appdir = c:\userapp
```

6. Change the application header file name in the following line:

```
appinc = $(appdir)\sample.h
```

   to:

```
appinc = $(appdir)\app.h
```

7. Change the source file names in the following lines. List `appsrc` must enumerate all application source files.

```
appsrc = \
$(appdir)\samplesr.c \
$(appdir)\samplepc.c
```

   to:

```
appsrc = \
$(appdir)\app1.c \
$(appdir)\app2.c
```

8. Change the object file names in the following lines. List `appobj` must enumerate all application object files.

```
appobj = \
$(object)\samplesr.obj \
$(object)\samplepc.obj
```

   to:

```
appobj = \
$(object)\app1.obj \
$(object)\app2.obj
```

NOTE If the application has more than two source files, you have to add files in 'appsrc' and 'appobj' lists. You can also leave one filename in each list if the application has one source file only.

9. Change the OIL file name in the following line:

```
oilname = cfg555db
```

   to:

```
oilname = appcfg
```

10. Change the executed binary file name in the following line:

```
exename = sample
```

   to:

```
exename = app
```

11. Create a batch file `mk.bat` in the directory `c:\userapp` and add the following lines to the file:

```
set CWDIR=c:\metrowerks\codewarrior
set DIABDIR=c:\diab
set OSEKDIR=c:\metrowerks\osek\osmpc
set SYSGENDIR=c:\metrowerks\osek
nmake
```

Make sure that you have placed actual paths to the CodeWarrior compiler, Diab Data compiler, OSEK OS and SysGen directories instead of the examples you can see above.

If you use the GNU make utility, you shall change the `nmake` command to the following lines:

```
set MAKE_MODE=unix
make
```

Slash can be used instead of backslash in the directory names.

The `mk.bat` file will be used to set the environment variables and to start operations defined in the makefile.

NOTE You can skip DIABDIR variable if you use CodeWarrior compiler only. You can skip CWDIR variable if you use Diab Data compiler only.

Running Application

To compile the created application and execute it take the following steps:

1. Open the command prompt window.
2. Change the current directory to `c:\userapp`.
3. Execute the command `mk`. After the application building has been completed, subdirectories `gen`, `obj` and `bin` are placed in `c:\userapp` directory. `Gen` includes files generated by the System Generator. `Obj` includes object files. Bin includes the executable file, ORTI file and memory map.

   If the application making has been completed successfully, the following files are created in the `userapp` directory:

   – `gen` subdirectory:

`appcfg.c` - system objects definition;

`appcfg.h` - system objects header file;

`osprop.h` - system properties;

– `obj` subdirectory:

`os.obj, osalm.obj, osctr.obj, osevt.obj, osisr.obj, osmsg.obj, osres.obj, ossch.obj, osset.obj, ostrg.obj and ostsk.obj` - OSEK OS object files;

`appcfg.obj` - object file for system objects;

`app1.obj` and `app2.obj` - application object files;

`crts.obj` or `__start.obj` - start up object file;

`vector.obj` - initialization code and interrupt vector table object file;

– `bin` subdirectory:

`app.elf` - executable file;

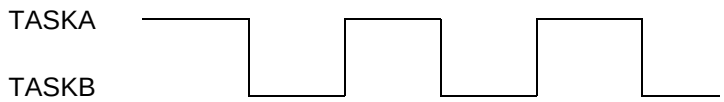
`app.map` - linker map of the application;

Some additional files such as assembler listings are also being created during the application making.

4. Start the debugger.
5. Load file `c:\userapp\bin\app.elf` into the debugger.
6. Set breakpoints at TASKA and TASKB functions.
7. Reset and run the application. The application shall break on TASKA and TASKB by rotation.

The application implements the following algorithm: TASKA is autostarted by the OS. This task activates TASKB which has a lower priority. Then TASKA terminates itself and the OS starts TASKB activated by TASKA. Then TASKB chains TASKA. Therefore TASKB terminates itself and TASKA is transferred to *running* state. It is the original position.

The scenario repeats endlessly. The diagram of task switching sequence is shown below.



To clean all built files perform one of the following actions:

- Delete subdirectories `gen`, `obj` and `bin`.
- Open the command prompt window. Set the current directory to `c:\userapp`. Execute nmake clean (for Microsoft NMAKE utility) or make clean (for GNU make utility) command.

Additional Task

This section describes how to add an additional task.

Configuration File

To add a task take the following steps:

1. Open `c:\userapp\appcfg.oil` file in a text editor.
2. To define a new task add the following statements to the end of `appcfg.oil` file (before closing brace for *CPU*).

```
TASK TASKC {
       PRIORITY = 3;
       SCHEDULE = FULL;
       AUTOSTART = TRUE;
       ACTIVATION = 1;
     };
```

   The task TASKC is autostarted. It has the highest priority. Therefore TASKC is the first started task and any other task can not interrupt the task TASKC.

3. Save `appcfg.oil` file.

Source Code

The next step is creation of the task TASKC source code. The only action of this function is terminating itself.

Take the following steps to modify the application code:

1. Open file `c:\userapp\app2.c` in a text editor.
2. To define the task TASKC add the following code to the end of file `app2.c`.

```
TASK( TASKC )
{
    TerminateTask();
}
```

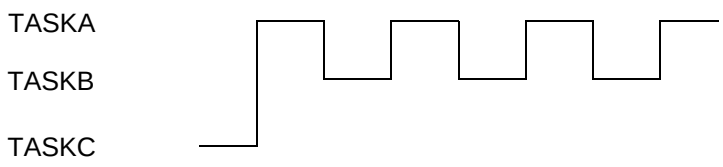
3. Save file `c:\userapp\app2.c`.

Running Application

To execute the application take the following steps:

1. Open the command prompt window.
2. Change the current directory to `c:\userapp`.
3. Execute the command `mk`. Files `mk.bat` and `makefile` have been created in "MakeFile". They have not been modified.
4. Start the debugger.
5. Load file `c:\userapp\bin\app.elf` into the debugger.
6. Set breakpoints at TASKA, TASKB and TASKC functions.
7. Reset and run the application. The application will break on the task TASKC. Then the application will break on the task TASKA and TASKB by rotation.

The diagram of task switching sequence is shown below.



Adding Single Alarm

This section contains a description how to add an *alarm* to the application. The system timer will be used to increment the counter attached to the alarm. The alarm will be set to a relative value by the task TASKC. Then TASKC terminates itself and TASKA is transferred to *running* state. When the alarm expires, it activates the task TASKC. TASKC has the highest priority, therefore it interrupts TASKA or TASKB. The task TASKC sets the alarm again and terminates itself. This process will repeat periodically. TASKA and TASKB are working in background.

Configuration File

To use a system timer, counter and alarm in the application corresponding objects shall be added to the OIL file. Take the following steps:

1. To define the System Timer we have to choose the hardware source of the timer interrupts which are handled by the OS and to define parameters to configure the period for the system timer. There are two types of the system timer supported by the OS - HWCOUNTER and SWCOUNTER. The HWCOUNTER has a less system overhead because the interrupts occur only if an alarm attached to the counter expires. But the HWCOUNTER does not use the whole set of the timer hardware sources. So the decision which type of the system timer to choose shall be based on the available hardware and application requirements. In this example we will use the HWCOUNTER with the period (tick duration) of 4 microsecond. To configure the system timer add the following statements into the OS section of the *OIL* file `appcfg.oil` between line *"TargetMCU* = MPC555 *{"* and *"};"* .

```
ClockFrequency=4000;
SysTimer = HWCOUNTER {
  COUNTER = TaskCounter;
  ISRPRIORITY = 8;
  Period = 4000;
    };
  TimerHardware = TB0 {
    Prescaler = OS;
    Freeze = TRUE;
  };
};
```

The Prescaler is automatically calculated by the System Generator. The timer modulo value is not used for the HWCOUNTER configuration.

2. To declare a counter for the System Timer add the following statements to the end of `appcfg.oil` file (before closing brace for CPU). The counter will be increased periodically on every System Timer tick.

```
COUNTER TaskCounter {
       MINCYCLE = 0;
       MAXALLOWEDVALUE = 0xFFFFFFFF;
       TICKSPERBASE = 10;
     };
```

3. To declare an alarm attached to the counter TaskCounter add the following statements to the end of `appcfg.oil` file (before closing brace for CPU). This alarm activates the task TASKC.

```
ALARM AL1 {
       COUNTER = TaskCounter;
       ACTION = ACTIVATETASK {
         TASK = TASKC;
       };
     };
```

Source Code

The task TASKC code must be changed. To provide the alarm setting add the following statements into the task TASKC code (file `c:\userapp\app2.c`) before *TerminateTask()*; statement: *SetRelAlarm* (AL1, 250, 0);

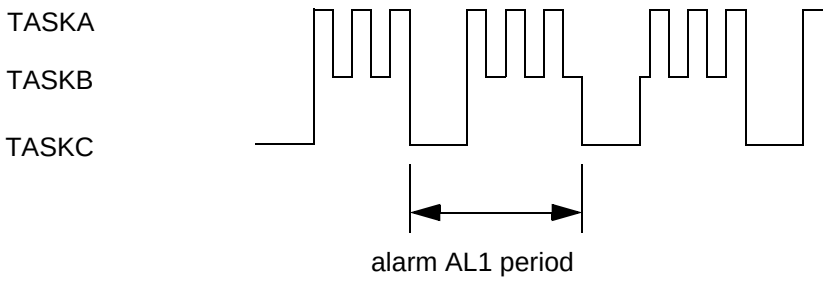
Running Application

To execute the application take the following steps:

1. Open the command prompt window, change the current directory to `c:\userapp` and execute command `mk`.
2. Start the debugger.
3. Load file `c:\userapp\bin\app.elf` into the debugger.
4. Set a breakpoint at TASKC function.

5. Reset and run the application. The application will break on the task TASKC periodically.
6. Add breakpoints at TASKA and TASKB functions.
7. Run the application again. You can see now that the task TASKC periodically interrupts the tasks TASKA and TASKB which call each other by rotation.

The diagram of task switching sequence is shown below.



A number of TASKA / TASKB activations between adjacent TASKC starts depends on CPU clock frequency and can differ from the number shown at the diagram.

It can happen that the TASKA / TASKB execution cycle and the alarm cycle do not have a common multiple. Therefore the number of TASKA and TASKB activations can vary slightly in different TASKC executions.

Using Event and Extended Task

The periodically activated task was created in the previous section. Similar results can be achieved using an extended task and an event. It allows the application to avoid task restarting. The extended task will be autostarted and never terminated. The task will periodically activate a function with a period of alarm AL1. Between adjacent function calls the extended task will be transferred into *waiting* state.

Configuration File

The task TASKC will be used as extended task. To adjust the task and to add an event corresponding objects shall be prepared in the OIL file. Take the following steps:

1. Open `c:\userapp\appcfg.oil` file in a text editor.
2. Add a reference to the event to a TASKC object. Here is the corrected code of this object:

```
TASK TASKC {
       PRIORITY = 3;
       SCHEDULE = FULL;
       AUTOSTART = TRUE;
       ACTIVATION = 1;
       EVENT = Cycle;
       STACKSIZE = 256;
     };
```

   The task TASKC is autostarted. It has a reference to the event Cycle. Therefore it is an extended task. Existence of the extended task leads to *ECC1* Conformance Class which is selected automatically by the System Generator. The task TASKC has the highest priority. Therefore any other task can only preempt TASKC if the task TASKC is terminated or transferred to *waiting* state.

3. Since the TASKC is an extended task which has a separate stack and there is a System Timer ISR (category 2), therefore the ISR stack must be defined. Add the following statement to the OS section:

```
IsrStackSize = 256;
```

4. According to the new scenario the alarm will not activate the task TASKC. The alarm will set an event "Cycle" for the task TASKC. Change the object AL1 definition according to the following pattern:

```
ALARM AL1 {
       COUNTER = TaskCounter;
       ACTION = SETEVENT {
         TASK = TASKC;
         EVENT = Cycle;
       };
     };
```

5. To define an event for the task TASKC add the following statement to the end of `appcfg.oil` file (before closing brace for CPU). Mask of the event is calculated automatically by the System Generator.

```
EVENT Cycle { MASK = AUTO; };
```

6. Save `appcfg.oil` file.

Source Code

The task TASKC will periodically call a function CycleFunc. The only action of this function is to increment a variable Counter. In a practical application the function can perform other actions.

Take the following steps to modify the application code:

1. Open `c:\userapp\app2.c` file in a text editor.
2. Add the Counter variable declaration to the beginning of the file `app2.c`.

```
int Counter;
```

3. Modify the task TASKC according to the following template:

```
TASK( TASKC )
{
     Counter = 0;
     while( 1 )
     {
       SetRelAlarm ( AL1, 250, 0 );
       WaitEvent( Cycle );
       CycleFunc();
       ClearEvent( Cycle );
     }
     TerminateTask();
}
```

4. Add the function CycleFunc before the task TASKC definition:

```
void CycleFunc( void )
{
     Counter++;
}
```

The task TASKC performs the following actions in the application:

1. The task TASKC is autostarted by the OS.
2. The task clears Counter.
3. The task runs an infinite loop.
4. The first step of the loop is setting a relative alarm AL1 which expires after 250 ticks of the counter TaskCounter (after `1` ms).
5. Then *WaitEvent* service is called. The task is transferred by this service to *waiting* state and keeps in this state until the alarm expires. Another task can be running while TASKC waits for the next alarm AL1 expiration.
6. The function CycleFunc is called at the next step of the loop. Therefore the function is called after each alarm AL1 has expired.
7. The last step of the loop is clearing the event in order to allow the *waiting* state at the next loop. Then the task jumps to step 4 and repeats steps 4–7.

This algorithm causes periodical calling of the function CycleFunc every 250 ticks of the counter TaskCounter (every `1` ms).

The task TASKC shares CPU time with the TASKA and TASKB tasks which call each other. The TASKC task's priority is higher than the TASKA and TASKB tasks' ones. Therefore TASKC interrupts TASKA or TASKB execution.

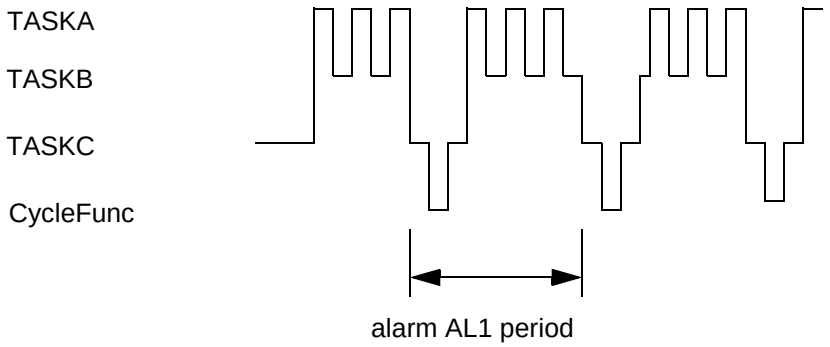
Running Application

To execute the application take the following steps:

1. Open the command prompt window, change the current directory to `c:\userapp` and execute the command `mk`.
2. Start the debugger.
3. Load file `c:\userapp\bin\app.elf` into the debugger.
4. Set a breakpoint to the found address of the CycleFunc function.
5. Reset and run the application. The application will break on the function CycleFunc periodically. The value Counter is increased on every break.
6. Add breakpoints at TASKA and TASKB functions. You can see now that the CycleFunc function periodically interrupts the TASKA and TASKB tasks which call each other by rotation. The

TASKC task is preempted while waiting for the alarm AL1 expiration. The TASKC task's priority is highest, therefore the OS returns operation to the TASKC task straight after the alarm has expired and sets the event Cycle.

The diagram of task switching sequence is shown below.



A number of TASKA / TASKB activations between adjacent *CycleFunc* calls depends on CPU clock frequency and can differ compared to the number shown at the diagram.

It can happen that the TASKA / TASKB execution cycle and the alarm cycle do not have a common multiple. Therefore the number of TASKA and TASKB activations can vary slightly in different CycleFunc executions.

Cyclic Alarm

The Cyclic alarm can be used instead of periodically setting of the single alarm. It allows a more accurate controlling the period. This section describes how to change a single alarm to a cycle one and to keep the previous functionality.

There is no need to modify OIL file. Only the TASKC source code will be corrected.

Source Code

To use the cyclic alarm instead of a single one take the following steps:

1. Open `c:\userapp\app2.c` file in a text editor.
2. Correct the TASKC code according to the following template:

```
TASK( TASKC )
{
     Counter = 0;
     SetRelAlarm ( AL1, 250, 250 );
     while( 1 )
     {
       WaitEvent( Cycle );
       CycleFunc();
       ClearEvent( Cycle );
     }
     TerminateTask();
}
```

Now the task TASKC performs the following actions in the application:

1. The task TASKC is autostarted by the OS.
2. The task clears Counter.
3. It sets a relative alarm AL1 which expires periodically every 250 ticks of the counter TaskCounter (every 1 ms).
4. The task TASKC runs an infinite loop.
5. The first step of the loop is waiting for an event which transfers the task to *waiting* state and the task keeps in this state until the alarm expires. Another task can be running while the TASKC is waiting for the next alarm AL1 expiration.
6. The function CycleFunc is called at the next step of the loop. Therefore the function is called after each time when the alarm AL1 has expired.
7. The last step of the loop is clearing the event in order to allow transferring into *waiting* state at the next loop. Then the task jumps to step 5 and repeats steps 5–7.

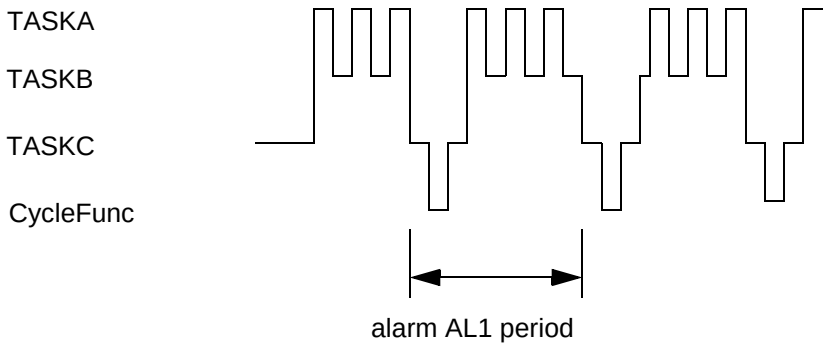
This algorithm causes periodical calling of the function CycleFunc every ticks of the counter TaskCounter.

The task TASKC shares CPU time with the TASKA and TASKB tasks which call each other. The task TASKC priority is higher than the TASKA and TASKB tasks' ones. Therefore TASKC interrupts TASKA or TASKB execution.

Running Application

To execute the application take the same steps as described in "Using Event and Extended Task".

The diagram of task switching sequence is shown below.



A number of TASKA / TASKB activations between adjacent CycleFunc calls depends on CPU clock frequency and can differ compared to the number shown at the diagram.

It can happen that the TASKA / TASKB execution cycle and the alarm cycle do not have a common multiple. Therefore the number of TASKA and TASKB activations can vary slightly in different CycleFunc executions.

TimeScale

The TimeScale is OSEKturbo extension of the OSEK OS. This mechanism allows the application to increase performance for set of periodic tasks' activations - it is a kind of a static schedule. The TimeScale mechanism can be used when the sequence of task activations of reasonable size can be defined. For example, there are three tasks in the application, TASK1, TASK2, and TASK3, each of the tasks has a period of 10 milliseconds and is executed in the following sequence: TASK2 starts 5 milliseconds later

than TASK1, and TASK3 starts 2 milliseconds later than TASK2. This sequence of task activations repeats each the period of 10 milliseconds. The TimeScale is attached to the system timer configured as HWCOUNTER, and no other alarms shall be attached to it. So we will configure the second timer to attach the TaskCounter.

The same application structure that used in the previous examples is a base for the next example.

Configuration File

The OIL file shall be changed to add new tasks, configure the TimeScale and the second timer. Take the following steps:

1. Open `c:\userapp\appcfg.oil` file in a text editor.
2. Create a definition for three new tasks:

```
TASK TASK1 {
       PRIORITY = 4;
       SCHEDULE = FULL;
       AUTOSTART = FALSE;
       ACTIVATION = 1;
     };

     TASK TASK2 {
       PRIORITY = 5;
       SCHEDULE = FULL;
       AUTOSTART = FALSE;
       ACTIVATION = 1;
     };
     TASK TASK3 {
       PRIORITY = 6;
       SCHEDULE = FULL;
       AUTOSTART = FALSE;
       ACTIVATION = 1;
     };
```

3. To configure the TimeScale the following statements shall be added to the OS section:

```
TimeScale = TRUE {
         TimeUnit = ms;
         Step = SET {
```

```
      StepNumber = 1;
      StepTime = 5;
      TASK = TASK1;
    };
    Step = SET {
      StepNumber = 2;
      StepTime = 2;
      TASK = TASK2;
    };
    Step = SET {
      StepNumber = 3;
      StepTime = 3;
      TASK = TASK3;
    };
  };
```

The configured TimeScale has three steps, at the first step *TASK1* starts, the second step is 5 milliseconds after the first step and *TASK2* is activated, the third step is 2 milliseconds after the second step and *TASK3* is activated, 3 milliseconds after the third step the TimeScale will execute the first step. All the time intervals for the TimeScale are configured in milliseconds - the "TimeUnit = ms;" statement allows the definition of time measurement units for the TimeScale, ticks of the System Timer are used by default.

4. To keep the existing application part which serves periodic event setting for TASKC, the TaskCounter with the attached alarm AL1 shall be reassigned to the second timer. The second timer definition shall be added to the OS section of the OIL file `appcfg.oil` between line *"TargetMCU = MPC555 {"* and corresponding closing bracket *"};"* below the SysTimer definition.

```
SecondTimer = SWCOUNTER {
  COUNTER = TaskCounter;
  ISRPRIORITY = 16;
  TimerHardware = DEC {
    Prescaler = OS {
      Value = 0;
    };
    TimerModuloValue = 600;
    Freeze = TRUE;
  };
};
```

The DEC is chosen as interrupt source for the second timer. The period of the timer is defined by the *Prescaler* setting which is controlled by the OS, the tick duration is calculated by the system generator, for the DEC timer with *Prescaler* `0` and frequency 4 MHz it is equal to 2400 microseconds.

5. Add the definition of the counter which is to be attached to the system timer to the end of `appcfg.oil` file (before closing brace for CPU).

```
COUNTER SystemTimer {
       MINCYCLE = 0;
       MAXALLOWEDVALUE = 0xFFFFFFFF;
       TICKSPERBASE = 10;
     };
```

Correct assigned *COUNTER* in SysTimer definition:

```
SysTimer = HWCOUNTER {
  COUNTER = SystemTimer;
  ...
};
```

6. Save `appcfg.oil` file.

Source Code

The next step is creation of the source code for new tasks. The only action of these tasks is terminating itself. To activate the TimeScale the *StartTimeScale* service shall be executed - add this functionality to TASKC.

Take the following steps to modify the application code:

1. Open file `c:\userapp\app2.c` in a text editor.
2. To define tasks TASK1, TASK2, TASK3 add the following code to the end of the file `app2.c`.

```
TASK( TASK1 )
{
     TerminateTask();
}
TASK( TASK2 )
{
     TerminateTask();
```

```
}
TASK( TASK3 )
{
     TerminateTask();
}
```

3. Add call of *StartTimeScale* service to TASKC after *ClearEvent* statements according to the following template:

```
TASK( TASKC )
{
  Counter = 0;                    /* initialize counter */
  SetRelAlarm ( AL1, 10, 10 );  /* Set cyclic alarm */
  WaitEvent( Cycle );             /* Wait alarm AL1 expiration*/
  StartTimeScale();               /* Start Time Scale */
  while( 1 )                      /* infinite loop */
  {
    ClearEvent( Cycle );/* Clear event */
    WaitEvent( Cycle );/* Wait alarm AL1 expiration*/
      /* Call CycleFunc when alarm set event */
    CycleFunc();
  }
                              /* This line is never reached */
}
```

4. Save file `c:\userapp\app2.c`.

The tasks TASK1, TASK2 and TASK3 are activated periodically and interrupt execution of TASKA or TASKB or TASKC.

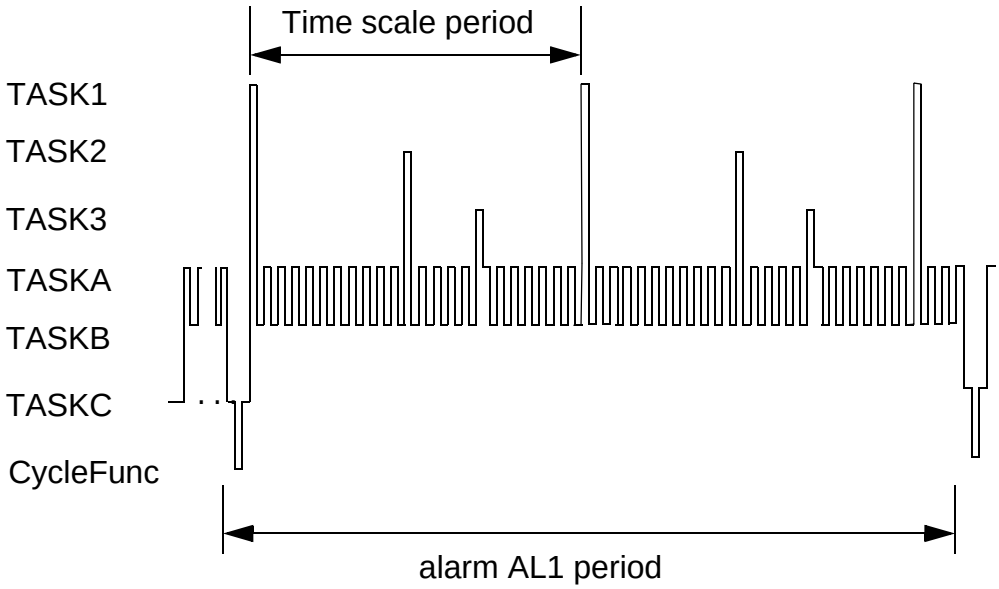
Running Application

To execute the application take the following steps:

1. Open the command prompt window, change the current directory to `c:\userapp` and execute the command `mk`.
2. Start the debugger.
3. Load file `c:\userapp\bin\app.` into the debugger.
4. Set breakpoints at TASK1, TASK2, TASK3 and CycleFunc functions.
5. Reset and run the application. After the first break on *CycleFunc* the *TimeScale* will be started and control will be passed to TASK1 -

the first task in *TimeScale* activated immediately. Then the application will break on those functions periodically.

The diagram of task switching sequence is shown below.



A number of TASKA / TASKB activations between adjacent CycleFunc calls depends on CPU clock frequency and can differ compared to the number shown at the diagram.

Listing

You can find below a complete listing of the updated source files. This listing corresponds to the application described in "TimeScale".

File `appcfg.oil`:

```
OIL_VERSION = "2.3";
#include "ost22mpc.oil"
CPU cpu1 {
    APPMODE Mode {};
    OS os1 {
        STATUS = EXTENDED;
        TargetMCU = MPC555 {
```

```
        ClockFrequency=4000;
        SysTimer = HWCOUNTER {
            COUNTER = TaskCounter;
            ISRPRIORITY = 8;
            Period = 4000;
                };
        };
        SecondTimer = SWCOUNTER {
            COUNTER = TaskCounter;
            ISRPRIORITY = 16;
            TimerHardware = DEC {
                Prescaler = OS {
                    Value = 0;
                };
                TimerModuloValue = 600;
                Freeze = TRUE;
            };
        };
    };
    TimeScale = TRUE {
        TimeUnit = ms;
        Step = SET {
            StepNumber = 1;
            StepTime = 5;
            TASK = TASK1;
        };
        Step = SET {
            StepNumber = 2;
            StepTime = 2;
            TASK = TASK2;
        };
        Step = SET {
            StepNumber = 3;
            StepTime = 3;
            TASK = TASK3;
        };
    };
    IsrStackSize = 256;
    STARTUPHOOK = FALSE;
    SHUTDOWNHOOK = FALSE;
    PRETASKHOOK = FALSE;
    POSTTASKHOOK = FALSE;
```

```
    ERRORHOOK = FALSE;
    USEGETSERVICEID = FALSE;
    USEPARAMETERACCESS = FALSE;
};

TASK TASKA {
    PRIORITY = 2;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
};
TASK TASKB {
    PRIORITY = 1;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
};
TASK TASKC {
    PRIORITY = 3;
    SCHEDULE = FULL;
    AUTOSTART = TRUE;
    ACTIVATION = 1;
    STACKSIZE = 256;
    EVENT = Cycle;
};

TASK TASK1 {
    PRIORITY = 4;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
};
TASK TASK2 {
    PRIORITY = 5;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
};
TASK TASK3 {
    PRIORITY = 6;
    SCHEDULE = FULL;
    AUTOSTART = FALSE;
```

```
        ACTIVATION = 1;
    };

    COUNTER SystemTimer {
        MINCYCLE = 0;
        MAXALLOWEDVALUE = 0xFFFFFFFF;
        TICKSPERBASE = 10;
    };
    COUNTER TaskCounter {
        MINCYCLE = 0;
        MAXALLOWEDVALUE = 0xFFFFFFFF;
        TICKSPERBASE = 10;
    };
    ALARM AL1 {
        COUNTER = TaskCounter;
        ACTION = SETEVENT {
            TASK = TASKC;
            EVENT = Cycle;
        };
    };
    EVENT Cycle { MASK = AUTO; };
};
```

File `app1.c`:

```
#include "osprop.h"         /* OS Properties file */
#include <osapi.h>          /* OSEK API declarations */
#include "app.h"            /* application header */
#include <appcfg.h>         /* definitions for system objects */

int main( void )        /* entry point of the application */
{
    StartOS( Mode );    /* jump to OSEK startup */
}
TASK( TASKA )/* task A */
{
    ActivateTask( TASKB );    /* Activate task TASKB */
          /* TASKB priority is lower than TASKA priority */
          /* Therefore TASKB transfer to ready state by */
          /* ActivateTASK service */
    TerminateTask( );     /* TASKA Terminate itself */
          /* TASKB will be transferred to the running */
```

```
          /* state after terminating TASKA */
}
```

File `app2.c`:

```
#include "osprop.h"       /* OS Properties file */
#include <osapi.h>        /* OSEK API declarations */
#include "app.h"          /* application header */
#include <appcfg.h>       /* definitions for system objects */
int Counter;              /* CycleFunc entry counter */
TASK( TASKB )             /* task B */
{
    ChainTask( TASKA );   /* Chain to TASKA */
          /* TASKB is terminated by this service call */
          /* TASKA is activated as a chain task */
          /* TASKA will be transferred to the running */
          /* state after TASKB termination */
}
void CycleFunc( void )
{         /* This function is called periodically */
    Counter++;  /* Increment entry counter */
}
TASK( TASKC )
{
    Counter = 0;                  /* initialize counter */
    SetRelAlarm ( AL1, 10, 10 );  /* Set cyclic alarm */
    WaitEvent( Cycle );           /* Wait alarm AL1 expiration*/
    StartTimeScale();             /* Start Time Scale */
    while( 1 )                    /* infinite loop */
    {
      ClearEvent( Cycle );        /* Clear event */
      WaitEvent( Cycle );         /* Wait alarm AL1 expiration*/
                     /* Call CycleFunc when alarm set event */
      CycleFunc();
    }
                     /* This line is never reached */
}
TASK( TASK1 )
{
  TerminateTask();
}
TASK( TASK2 )
```

```
{
  TerminateTask();
}
TASK( TASK3 )
{
  TerminateTask();
}
```

File `app.h`:

```
#ifndef APP_H
#define APP_H
#endif /* APP_H */
```

5

Using an Unsupported Target Derivatives

The chapter contains recommendations for the OSEK OS adaptation to other derivatives.

The current version of the OSEK OS supports MPC555, MPC561, MPC563 and MPC565 MCUs directly.

The user can try to adapt the OSEK OS to other MPC derivatives. The OSEK OS will work correctly in most cases.

This chapter consists of the following sections:

Target MCU Type

If you want to use the OSEK OS with other derivatives set *TargetMCU* option to MPC. This value turns off derivative specific features which can cause some problems if the OSEK OS runs on an unsupported MCU. The main restriction is impossibility of the system and the second timers definition. *SysTimer* and *SecondTimer* blocks can not be defined in the OS section of the OIL file. Therefore the user should define a timer in an application (if the timer is needed). The following restrictions are also applicable for *TargetMCU* equal to MPC:

- *ClockFrequency*, *ClockDivider* and *ClockMultiplier* attributes are not applicable

If it is planned to use an unsupported MCU which structure is close to one of the MCUs supported by the OSEK OS, the *TargetMCU* attribute can be set to another value than MPC. It allows configuring timers by the System

Generator. If you try to use the value MPC555 with another derivative, please be very careful. Different derivatives can have different timer structures, memory map and vector table. There are the following recommendations for the case if you try to use another derivative and *TargetMCU* is not set to MPC:

- If the system (second) timer is used, check out that specified timer hardware is identical to the specified and actual derivatives. Both derivatives must have the same timer structure, equal timer register addresses and equal timer interrupt vector address.

Vector Table

If *TargetMCU* is set to MPC, the OSEK OS does not provide the vector table and the user should create a vector table corresponding to the CPU vectors. To create a vector table take the following steps:

1. Copy file `$OSEKDIR/hwspec/vector.c` to the application directory.
2. Open a new `vector.c` file in any text editor.

The `$OSEKDIR/hwspec/vector.c` file may be used as basis for new vector table.

3. Save and close the `vector.c` file.

You can find comments on using the `vector.c` file in "Source Code".

System Timer

If another derivative is used and the *TargetMCU* attribute is set to MPC, the OSEK OS does not provide a system timers. Note that the timer(s) with software counter only can be added. If the timer is required, it should be added to the user's application. The following steps describe how to implement a timer in the application code.

1. Add an *ISR* object definition to the OIL file. This ISR will be used as a system timer interrupt handler:

```
ISR UserTimerHandler {
     CATEGORY = 2;
     PRIORITY = 15;
};
```

2. Add COUNTER object to the OIL file. This counter will be increased by the *SysHandler* routine. The value of the counter attributes should be set according to the application algorithm. The values shown below are an example only.

```
COUNTER UserCounter {
     MINCYCLE = 3;
     MAXALLOWEDVALUE = 255;
     TICKSPERBASE = 10;
};
```

3. Create a function *InitializeTimer* in the application source file. This function should contain a code for hardware timer initialization and timer start up. The function should be called in the *StartupHook* or from the autostarted task.

```
void InitializeTimer() {
  /* initialize timer hardware registers */
  /* enable interrupts from the timer */
  /* start the timer */
}
```

4. If it is planned to use *ShutdownOS* service, then create a function *ShutdownTimer*. This function should contain a code for switching off the hardware timer and disabling timer interrupts. The function should be called after *ShutdownOS* service calling or in the *ShutdownHook*.

```
void ShutdownTimer() {
  /* disable interrupts from the timer */
  /* stop the timer */
  /* reset timer hardware registers */
}
```

5. Create ISR category 2 and add the following code to it. This ISR will be used as a timer interrupt handler.

```
ISR( SysHandler ){
  /* This interrupt handler is called by hardware */
  /* timer on every timer tick. */
  /* If it is need to correct timer operation, */
  /* modify hardware registers here */
```

```
  CounterTrigger(SysCounter);
}
```

6. Modify the vector table to support the ISR.

Make File

It is recommended to use the makefile from the OSEK OS sample to compile an application (see “MakeFile”). The makefile has to be corrected if the application is compiled for another derivative. Select the most appropriate makefile in the `sample` subdirectory and copy the makefile to the application directory. Then correct the following parameters in the created makefile:

- application dependent names (see “MakeFile”)
- *intmembase* value to map RAM according to the CPU memory map in the “MEMORY” part of linker script

A

Quick Reference

The appendix contains lists of OSEK OS run-time services with entry and exit conditions as well as OIL object parameters with their possible values and short descriptions.

This appendix consists of the following sections:

System Services Quick Reference

The list of all OSEK Operating System run-time services is provided below. Input and output parameters, syntax and ability to use by OSEK entities are shown. Note that ISR means ISR category 2 if not specified else

Table A.1 OSEK OS Services

Service	Input	Output	Allowed In
Task management services			
ActivateTask	Task name	–	Task, ISR
	syntax: StatusType ActivateTask(TaskType <TaskID>);		
TerminateTask	–	–	Task
	syntax: StatusType TerminateTask(void);		
ChainTask	Task name	–	Task
	syntax: StatusType ChainTask(TaskType <TaskID>);		
Schedule	–	–	Task
	syntax: StatusType Schedule(void);		
GetTaskId	–	Task name	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetTaskId(TaskRefType <TaskIDRef>);		

Table A.1 OSEK OS Services

Service	Input	Output	Allowed In
GetTaskState	Task name	Task state	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetTaskState(TaskType <TaskID>, TaskStateRefType <StateRef>);		
Interrupt management services			
EnableAllInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void EnableAllInterrupts(void);		
DisableAllInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void DisableAllInterrupts(void);		
ResumeAllInterrupts	–	–	Task, ISR category 1 and 2, alarm-callbacks
	syntax: void ResumeAllInterrupts(void);		
SuspendAllInterrupts	–	–	Task, ISR category 1 and 2, alarm-callbacks
	syntax: void SuspendAllInterrupts(void);		
ResumeOSInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void ResumeOSInterrupts(void);		
SuspendOSInterrupts	–	–	Task, ISR category 1 and 2
	syntax: void SuspendOSInterrupts(void);		
Resource management services			
GetResource	Resource name	–	Task, ISR
	syntax: StatusType GetResource(ResourceType <ResID>);		
ReleaseResource	Resource name	–	Task, ISR
	syntax: StatusType ReleaseResource(ResourceType <ResID>);		

Table A.1 OSEK OS Services

Service	Input	Output	Allowed In
Event control services			
SetEvent	Taks name, Event mask	–	Task, ISR
	syntax: StatusType SetEvent (TaskType <TaskID>, EventMaskType <Mask>);		
ClearEvent	Event mask	–	Extended task
	syntax: StatusType ClearEvent(EventMaskType <Mask>);		
GetEvent	Task name	Event mask	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetEvent(TaskType <TaskID>, EventMaskRefType <Event>);		
WaitEvent	Event mask	–	Extended task
	syntax: StatusType WaitEvent(EventMaskType <Mask>);		
Counter management services			
InitCounter	Counter name, initial value	–	Task
	syntax: StatusType InitCounter(CtrRefType <CounterID>, TickType <Ticks>);		
CounterTrigger	Counter name	–	Task, ISR
	syntax: StatusType CounterTrigger(CtrRefType <CounterID>);		
GetCounterValue	Counter name	Counter value	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetCounterValue(CtrRefType <CounterID>, TickRefType <TicksRef>);		
GetCounterInfo	Counter name	Counter constants	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetCounterInfo(CtrRefType <CounterID>, CtrInfoRefType <InfoRef>);		

Table A.1 OSEK OS Services

Service	Input	Output	Allowed In
Alarm management services			
GetAlarmBase	Alarm name	Alarm constants	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetAlarmBase(AlarmType <AlarmID>, AlarmBaseRefType <InfoRef>);		
GetAlarm	Alarm name	Relative value in ticks before the alarm expires	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: StatusType GetAlarm(AlarmType <AlarmID>, TickRefType <TicksRef>);		
SetRelAlarm	Alarm name, Counter relative value, Cycle value	–	Task, ISR
	syntax: StatusType SetRelAlarm (AlarmType <AlarmID>, TickType <Increment>, TickType <Cycle>);		
SetAbsAlarm	Alarm name, Counter absolute value, Cycle value	–	Task, ISR
	syntax: StatusType SetAbsAlarm (AlarmType <AlarmID>, TickType <Start>, TickType <Cycle>);		
CancelAlarm	Alarm name	–	Task, ISR
	syntax: StatusType CancelAlarm(AlarmType <AlarmID>);		
<AlarmCallBack>[a]	–	–	–
	syntax: ALARMCALLBACK(<CallbackName>);		
StartTimeScale	–	–	Task
	syntax: void StartTimeScale(void);		
StopTimeScale	–	–	Task, ISR, all hook routines
	syntax: void StopTimeScale(void);		
Message management services			
SendMessage	Message name, message data	–	Task (all messages), ISR (unqueued WithCopy)
	syntax: StatusType SendMessage(SymbolicName <Message>, AccessNameRef <Data>);		

Table A.1 OSEK OS Services

Service	Input	Output	Allowed In
ReceiveMessage	Message name	Message data	Task (all types), ErrorHook (unqueued WithCopy), ISR (unqueued WithCopy)
	syntax: StatusType ReceiveMessage(SymbolicName <Message>, AccessNameRef <Data>);		
GetMessageResource	Message name	–	Task (WithoutCopy)
	syntax: StatusType GetMessageResource (SymbolicName <Message>);		
ReleaseMessageResource	Message name	–	Task (WithoutCopy)
	syntax: StatusType ReleaseMessageResource (SymbolicName <Message>);		
GetMessageStatus	Message name	–	Task (all types)
	syntax: StatusType GetMessageStatus (SymbolicName <Message>);		
InitCOM	–	–	–
	syntax: StatusType InitCOM (void);		
CloseCOM	–	–	–
	syntax: StatusType CloseCOM (void);		
StartCOM	–	–	Task (all types)
	syntax: StatusType StartCOM (void);		
StopCOM	–	–	–
	syntax: StatusType StopCOM (Scalar <ShutdownMode>);		
MessageInit	–	–	–
	syntax: StatusType MessageInit (void);		
ReadFlag	Flag name	–	–
	syntax: FlagValue ReadFlag (FlagType <FlagName>);		
ResetFlag	Flag name	–	–
	syntax: StatusType ResetFlag (FlagType <FlagName>);		
<MessageCallBack>[b]	–	–	–
	syntax: void <CallbackName> (void);		

Table A.1 OSEK OS Services

Service	Input	Output	Allowed In
Debugging services			
GetRunningStackUsage	–	–	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: unsigned short GetRunningStackUsage(void);		
GetStackUsage	Task name	–	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: unsigned short GetStackUsage(TaskType <TaskID>);		
GetTimeStamp	–	–	Task, ISR, ErrorHook, PreTaskHook, PostTaskHook
	syntax: unsigned short GetTimeStamp (void);		
Execution control services			
GetActiveApplicationMode	–	Current application mode	Task, ISR, All hooks
	syntax: AppModeType GetActiveApplicationMode(void);		
StartOS	Application mode name	–	Outside of OS
	syntax: void StartOS(AppModeType <Mode>);		
ShutdownOS	Error code	–	Task, ISR, StartupHook, ErrorHook
	syntax: void ShutdownOS(StatusType <Error>);		
Hook Routines			
ErrorHook	Error code	–	–
	syntax: void ErrorHook(StatusType <Error>);		
PreTaskHook	–	–	–
	syntax: void PreTaskHook(void);		
PostTaskHook	–	–	–
	syntax: void PostTaskHook(void);		
StartupHook	–	–	–
	syntax: void StartupHook(void);		

Table A.1 OSEK OS Services

Service	Input	Output	Allowed In
ShutdownHook	Error code	–	–
	syntax: void ShutdownHook(StatusType <Error>);		
IdleLoopHook	–	–	–
	syntax: void IdleLoopHook(void);		

a. <AlarmCallBack> is the value of the ALARMCALLBACKNAME attribute defined in ALARM object. The user can have several alarm callback functions, one for each alarm defined in the OIL file.

b. <MessageCallBack> is the value of the CALLBACKNAME attribute defined in MESSAGE object. The user can have several message callback functions, one for each message defined in the OIL file.

NOTE InitCounter, CounterTrigger, GetCounterValue, GetCounterInfo, StartTimeScale, StopTimeScale, GetRunningStackUsage, GetStackUsage, and GetTimeStamp services and IdleLoopHook hook are not defined in the OSEK OS v.2.2 specification. This is OSEKturbo extension of the OSEK OS.

The list of macros for parameter access from *ErrorHook* routine is provided below.

Table A.2 OSEK Macros for ErrorHook

Macro	Return Value
OSErrorGetServiceId()	Service identifier
OSError_StartOS_Mode()	Application mode
OSError_ActivateTask_TaskID()	Task identifier
OSError_ChainTask_TaskID()	Task identifier
OSError_GetTaskState_TaskID()	Task identifier
OSError_GetResource_ResID()	Resource identifier
OSError_ReleaseResource_ResID()	Resource identifier
OSError_SetEvent_TaskID()	Task identifier
OSError_GetEvent_TaskID()	Task identifier
OSError_SendMessage_Message()	Message identifier
OSError_ReceiveMessage_Message()	Message identifier
OSError_GetMessageResource_Message()	Message identifier
OSError_ReleaseMessageResource_Message()	Message identifier

Table A.2 OSEK Macros for ErrorHook

Macro	Return Value
OSErrorGetServiceId()	Service identifier
OSError_StartOS_Mode()	Application mode
OSError_GetMessageStatus_Message()	Message identifier
OSError_GetAlarmBase_AlarmID()	Alarm identifier
OSError_GetAlarm_AlarmID()	Alarm identifier
OSError_SetRelAlarm_AlarmID()	Alarm identifier
OSError_SetAbsAlarm_AlarmID()	Alarm identifier
OSError_CancelAlarm_AlarmID()	Alarm identifier
OSError_InitCounter_CounterID()[a]	Counter identifier
OSError_CounterTrigger_CounterID()[a]	Counter identifier
OSError_GetCounterValue_CounterID()[a]	Counter identifier
OSError_GetCounterInfo_CounterID()[a]	Counter identifier

a. Counter interface functions are not defined in OSEK OS v.2.2 specification, this is OSEKturbo extension of the OSEK OS.

The list of OSEK Operating System Data Types is provided here.

Table A.3 Data Types

Data Type	Description
AccessName	A unique name defining access to a message object
AccessNameRef	An address of the message data field
AlarmBaseRefType	The data type references data corresponding to the data type *AlarmBaseType*
AlarmBaseType	The data type represents a structure for storage of alarm characteristics. It is the same as *CtrInfoType*
AlarmType	The data type represents an alarm element
AppModeType	This data type represents the operating mode
CtrInfoRefType	The data type references data corresponding to the data type *CtrInfoType*
CtrInfoType	The data type represents a structure for storage of counter characteristics. This structure has the following fields: *maxallowedvalue* maximum possible allowed count value; *ticksperbase* number of ticks required to reach a counter-specific significant unit; *mincycle* minimum allowed number of ticks for a cyclic alarm (only for a system with Extended Status);

Table A.3 Data Types

Data Type	Description
CtrRefType	The data type references a counter
EventMaskRefType	The data type to refer to an event mask
EventMaskType	The data type of an event mask
FlagType	The data type of a message flag
ResourceType	The abstract data type for referencing a resource
StatusType	The data type for all status information the API services offer
SymbolicName	A unique name representing a message
TaskRefType	The data type to refer variables of the *TaskType* data type
TaskStateRefType	The data type to refer variables of the *TaskStateType* data type
TaskStateType	The data type for variables to store the state of a task
TaskType	The abstract data type for task identification
TickRefType	The data type references data corresponding to the data type *TickType*
TickType	The data type represents a counter value in system ticks

NOTE CtrRefType, CtrInfoType and CtrInfoRefType data types are not defined in the OSEK OS v.2.2 specification. This is OSEKturbo extension of the OSEK OS.

The list of OSEK Operating System constructional elements is provided below. All declarations are dummy, they are defined for compatibility with previous OSEK versions.

Table A.4 Constructional Elements

Name	Syntax
DeclareTask	DeclareTask(<name of task>)
DeclareISR	DeclareISR(<name of ISR>)
DeclareResource	DeclareResource(<name of resource>)
DeclareEvent	DeclareEvent(<name of event>)
DeclareCounter	DeclareCounter(<name of counter>)
DeclareAlarm	DeclareAlarm(<name of alarm>)

The table below contains all return values for the OSEK Operating System run-time services and error values.

Table A.5 Services Return and Error Values

Name	Value	Type
E_OK	0	No error, successful completion
E_OS_ACCESS	1	Access to the service/object denied
E_OS_CALLEVEL	2	Access to the service from the ISR is not permitted
E_OS_ID	3	The object ID is invalid
E_OS_LIMIT	4	The limit of services/objects exceeded
E_OS_NOFUNC	5	The object is not used, the service is rejected
E_OS_RESOURCE	6	The task still occupies the resource
E_OS_STATE	7	The state of the object is not correct for the required service
E_OS_VALUE	8	A value outside of the admissible limit
E_OS_SYS_STACK[a]	17	Task stack overflow
E_OS_SYS_ORDER [a]	18	Incorrect order of function calling
E_OS_SYS_MAINSTACK[a]	19	Main stack overflow
E_OS_SYS_ISRSTACK[a]	20	ISR stack overflow
E_COM_BUSY	33	Message in use by application task/function
E_COM_ID	35	Invalid message name passed as parameter
E_COM_LIMIT	36	Overflow of FIFO associated with queued messages
E_COM_LOCKED	39	Rejected service call, message object locked due to a pending operation
E_COM_NOMSG	41	No message available

[a.] E_OS_SYS_STACK is not defined in the OSEK OS v.2.2 specification. This is OSEKturbo extension of the OSEK OS.

The list of service identifiers for *ErrorHook* is provided below:

- identifieirs for standard OSEK services
  OSServiceId_StartOS
  OSServiceId_ShutdownOS
  OSServiceId_GetActiveApplicationMode
  OSServiceId_ActivateTask

OSServiceId_TerminateTask
OSServiceId_ChainTask
OSServiceId_Schedule
OSServiceId_GetTaskID
OSServiceId_GetTaskState
OSServiceId_ResumeAllInterrupts
OSServiceId_SuspendAllinterrupts
OSServiceId_ResumeOSInterrupts
OSServiceId_SuspendOSinterrupts
OSServiceId_EnableAllInterrupts
OSServiceId_DisableAllInterrupts
OSServiceId_GetResource
OSServiceId_ReleaseResource
OSServiceId_SetEvent
OSServiceId_ClearEvent
OSServiceId_GetEvent
OSServiceId_WaitEvent
OSServiceId_SendMessage
OSServiceId_ReceiveMessage
OSServiceId_GetMessageResource
OSServiceId_ReleaseMessageResource
OSServiceId_GetMessageStatus
OSServiceId_StartCOM
OSServiceId_StopCOM
OSServiceId_InitCOM
OSServiceId_CloseCOM
OSServiceId_GetAlarmBase
OSServiceId_GetAlarm
OSServiceId_SetRelAlarm
OSServiceId_SetAbsAlarm
OSServiceId_CancelAlarm

- identifiers for OSEKturbo specific services
  OSServiceId_InitCounter
  OSServiceId_CounterTrigger
  OSServiceId_GetCounterValue
  OSServiceId_GetCounterInfo
  OSServiceId_StartTimeScale
  OSServiceId_StopTimeScale

- identifier returned if the error occured not in the OS service called by the user but inside OS dispatcher
  OSServiceId_NoService

The following table contains OSEK Operating System constants with short descriptions.

Table A.6 OSEK OS Constants

Constant	Value	Description
RUNNING	0	Constant of data type *TaskStateType* for task state *running*
WAITING	1	Constant of data type *TaskStateType* for task state *waiting*
READY	2	Constant of data type *TaskStateType* for task state *ready*
SUSPENDED	3	Constant of data type *TaskStateType* for task state *suspended*
INVALID_TASK	Depends on user's settings in configuration OIL file 0	Constant of data type *TaskType* for a not defined task
RES_SCHEDULER		Constant of data type *ResourceType* for *Scheduler* as a resource
OSMAXALLOWEDVALUE		Maximum possible allowed system counter value
OSMAXALLOWEDVALUE2		Maximum possible allowed second counter value
OSTICKSPERBASE		Number of ticks required to reach a counter-specific value in the system counter
OSTICKSPERBASE2		Number of ticks required to reach a counter-specific value in the second counter
OSTICKDURATION		Duration of the system counter tick in nanoseconds
OSTICKDURATION2		Duration of the second counter tick in nanoseconds
OSMINCYCLE		Minimum allowed number of ticks for a cyclic alarm attached to the system counter (only for a system with Extended Status)
OSMINCYCLE2		Minimum allowed number of ticks for a cyclic alarm attached to the second counter (only for a system with Extended Status)
OSDEFAULTAPPMODE		Default application mode. This constant is always a valid parameter for *StartOS* service

Table A.6 OSEK OS Constants

Constant	Value	Description
OsBuildNumber	Current build number	Constant of data type (unsigned char*) which points to C-like NULL terminated string which contains the current build number. For example: 2.1.1.20

NOTE OSMAXALLOWEDVALUE2, OSTICKSPERBASE2, OSTICKDURATION2, OSMINCYCLE2 and OsBuildNumber constants are not defined in the OSEK OS v.2.2 specification. This is OSEKturbo extension of the OSEK OS.

OIL Language Quick Reference

The lists of all the OIL object parameters with their possible values and short descriptions are provided here. All standard object attributes must be always defined. OSEKturbo specific attributes can be defined in addition to standard ones. The value used by default is typed in boldface in the *Possible Values* cells.

Memory consumption and performance trends based on influence of individual attributes are signed in the *Possible Values* cells. There are three signs put next to the attribute values (exclude default value). They display variation of RAM usage, ROM usage and execution TIME (first, second and third sign respectively) compared to the default attribute value. Symbol “+” corresponds to increasing RAM, ROM or TIME, Symbol “–” corresponds to decreasing RAM, ROM and TIME and symbol “±” designates “no change”.

OS Object

The OS object is the mandatory one for any application. It defines the OS and its properties for the application. The OS attributes exactly correspond to the system options and are divided into parts corresponding to appropriate system objects. The standard and OSEKturbo specific

attributes of the OS object are marked by the "standard" and "specific" respectively.

Table A.7 OS Parameters

Object Parameters	Possible Values	Description
Global System Attributes		*This group of OS attributes represents system features which are common for the whole system*

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:
`STATUS = <STANDARD / EXTENDED>;`
`CC = <BCC1 / ECC1 / AUTO>;`
`DEBUG_LEVEL = <0 / 1 / 2 / 4>;`
`BuildNumber = <TRUE / FALSE>;`
`FastTerminate = <TRUE / FALSE>;`
`FastScheduler = <TRUE / FALSE>;`
`MessageCopyAllocation = <USER /OS>;`
`ResourceScheduler = <TRUE / FALSE>;`		

STATUS standard	STANDARD, EXTENDED (+,+,+)	This standard attribute specifies OS debug status
CC specific	BCC1, ECC1, **AUTO**	Specifies OSEK Conformance Class
DEBUG_LEVEL specific	**0** 1 (+,+,±) 2 (+,+,+) 4 (+,+,+)	Specifies the ORTI support in OS
BuildNumber specific	**TRUE** FALSE (±,-,±)	Specifies whether build number in ASCII form should be incorporated into OS binary image (ROM code) or not
FastTerminate specific	TRUE (-,-,-) **FALSE**	Specifies whether the fast version of Terminate/ ChainTask is used in BCC1 class
FastScheduler specific	TRUE (-,-,-) **FALSE**	Specifies whether the OS places global variables into CPU general purpose registers
MessageCopyAllocation specific	USER (±,±,±) **OS**	Specifies whether the System Generator generates copies of messages in global memory or message copies are allocated by the user
ResourceScheduler specific	**TRUE** FALSE(-,-,±)	Specifies whether RES_SCHEDULER should be supported or not

Table A.7 OS Parameters

Object Parameters	Possible Values	Description
CPU Related Attributes		*This group of OS attributes provides possibility to tune the selected hardware*

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:

```
TargetMCU = <name of MCU> {
      ClockFrequency = <integer / 4000>;
      ClockDivider = <integer / 1>;
      ClockMultiplier = <integer / 1>;
      SysTimer = <HWCOUNTER / SWCOUNTER / NONE> {
        COUNTER = <name of COUNTER>;
        ISRPRIORITY = <integer>;
        Period = <integer / AUTO>;
        TimerHardware = <name of timer hardware> {
          Prescaler = <USER / OS> {
            Value = <integer / AUTO>;
          };
          TimerModuloValue = <integer / AUTO>;
          Freeze = <TRUE / FALSE>;
          RTSEL = <EXTCLK / OSCM>;
        };
      };
      SecondTimer = <HWCOUNTER / SWCOUNTER / NONE> {
        COUNTER = <name of COUNTER>;
        ISRPRIORITY = <integer>;
        Period = <integer / AUTO>;
        TimerHardware = <name of timer hardware> {
          Prescaler = <USER / OS> {
            Value = <integer / AUTO>;
          };
          TimerModuloValue = <integer / AUTO>;
          Freeze = <TRUE / FALSE>;
          RTSEL = <EXTCLK / OSCM>;
        };
      };
      HCLowPower = <TRUE / FALSE>;
};
```

TargetMCU specific	MPC555, MPC561, MPC563, MPC565, MPC	Specifies target MCU type

Table A.7 OS Parameters

Object Parameters	Possible Values	Description
ClockDivider specific	integer	Specifies PLL divider (DIVF+1) for calculating input timer frequency. The attribute is applicable if the *TimerHardware* attribute has DEC, TB0 or TB1 value
ClockMultiplier specific	integer	Specifies PLL multiplier (MF+1) for calculating input timer frequency. The attribute is applicable if the *TimerHardware* attribute has DEC, TB0 or TB1 value
SysTimer specific	HWCOUNTER SWCOUNTER **NONE**	Defines whether the internal OS system timer is used or not. The attribute can not be defined if TargetMCU is set to MPC
SecondTimer specific	HWCOUNTER SWCOUNTER **NONE**	Defines whether the internal OS second timer is used or not. The attribute can not be defined if TargetMCU is set to MPC
COUNTER specific	name of COUNTER	Specifies the COUNTER which shall be attached to the system or second timer. The same counter can not be attached to the System and Second timers
Period specific	integer **AUTO**	Specifies period of a tick of the system (second) counter in nanoseconds
TimerHardware specific	PIT, DEC, RTC, TB0, TB1	The attribute is intended to select the hardware interrupt source for the system and second counters. The TimerHardware attributes in SysTimer and SecondTimer blocks can not have the same value
Prescaler specific	USER OS	Specifies whether prescaler value shall be initialized during OS startup or it is set by the user's code
Value (in Prescaler) specific	integer, **AUTO**	Defines initial prescaler value. Note that this attribute value is not equal to divide factor of timer hardware
TimerModuloValue specific	integer, **AUTO**	Specifies timer hardware register value
Freeze specific	TRUE FALSE	Defines whether timer stops or not if breakpoint is achieved in the debugger
RTSEL specific	EXTCLK OSCM	Specifies clock source for RTC and PIT timers
HCLowPower specific	TRUE (±,+,-) **FALSE**	Defines that low power mode shall be used when there are no ready or running tasks
ISRSourceControl specific	TRUE (+,+,+) **FALSE**	Defines that OS uses IMR registers in Suspend/ ResumeOSinterrupts functions

Table A.7 OS Parameters

Object Parameters	Possible Values	Description
Stack Related Attributes		*This group of OS attributes defines stack support in the system*

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:
`IsrStackSize = <integer>;`
`StackOverflowCheck = <TRUE / FALSE>;`		

StackOverflowCheck specific	TRUE (+,+,+) **FALSE**	Turns on stack overflow runtime checking and stack usage services
Task Related Attributes		This group of OS attributes controls task feature

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:
`TimeScale = <TRUE / FALSE> {`
`ScalePeriod = <integer / AUTO>;`
`TimeUnit = <ticks / ns / us / ms>;`
`Step = <SET> {`
`StepNumber = <integer>;`
`StepTime = <integer>;`
`TASK = <name of TASK>;`
`};`
`};`		

TimeScale specific	TRUE **FALSE**	Enables Time Scale mechanism
ScalePeriod specific	integer **AUTO**	Specifies full period of time scale in chosen measurement units
TimeUnit specific	**ticks**, ns, us, ms	Specifies measurement units: ticks means ticks of System Timer, ns means nanoseconds, us - microseconds, and ms - milliseconds
Step specific	SET	Defines one of step elements in the Time Scale
StepNumber specific	integer	Specifies the order of steps
StepTime specific	integer	Specifies the time until the next task activation in measurement units chosen by means of the TimeUnit attribute
TASK specific	name of TASK	Specifies the task to be activated

Table A.7 OS Parameters

Object Parameters	Possible Values	Description
Interrupt Related Properties		*This group of OS attributes defines parameters of ISR execution*

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:
`UnorderedExceptions = <TRUE / FALSE>;`
`InterruptDispatcher = <None / OneLevel / MultiLevel / Enhanced>;`		

UnorderedExceptions specific	TRUE (+,+,+) FALSE	Specifies whether unordered exception handling is supported or not
InterruptDispatcher specific	**None**, OneLevel, MultiLevel, Enhanced	Specifies interrupt mechanism
Hook Routines Related Attributes		*This group of OS attributes defines additional hook routines support in the system*

The attributes should be defined inside the scope of the OS object in accordance with the following syntax:
`STARTUPHOOK = <TRUE / FALSE>;`
`SHUTDOWNHOOK = <TRUE / FALSE>;`
`PRETASKHOOK = <TRUE / FALSE>;`
`POSTTASKHOOK = <TRUE / FALSE>;`
`ERRORHOOK = <TRUE / FALSE>;`
`USEGETSERVICEID = <TRUE / FALSE>;`
`USEPARAMETERACCESS = <TRUE / FALSE>;`
`IdleLoopHook = <TRUE / FALSE>;`		

STARTUPHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether StartupHook is called after the operating system starting up and before the dispatcher starting or not
SHUTDOWNHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether ShutdownHook is called during the system shutdown or not
PRETASKHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether PreTaskHook is called from the scheduler code before the operating system enters context of the task or not
POSTTASKHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether the PostTaskHook is called from the scheduler code after the operating system leaves the context of the task or not
ERRORHOOK standard	TRUE (±,+,+) FALSE	This standard attribute defines whether the ErrorHook is called by the system at the end of each system service which returns the status not equal to E_OK or not

Table A.7 OS Parameters

Object Parameters	Possible Values	Description
USEGETSERVICEID standard	TRUE (+,+,+) FALSE	Specifies ability of usage the access macros to the service ID in the error hook
USEPARAMETERACCESS standard	TRUE (+,+,+) FALSE	Specifies ability of usage the access macros to the context related information in the error hook
IdleLoopHook specific	TRUE (±,+,+) **FALSE**	Defines whether the IdleLoopH hook is called by the system from the scheduler idle loop (when there are no tasks in ready or running state) or not
Floating Point Related Attributes		*These are additional OS attributes to work with floating point*
The attributes should be defined inside the scope of the OS object in accordance with the following syntax: `FloatingPoint = <TRUE / FALSE> ;`		
FloatingPoint specific	FALSE TRUE (+,+,+)	Turns floating-point support

NOTE The *IdleLoopHook* hook is not defined in the OSEK OS v.2.2 specification. This is OSEKturbo extension of the OSEK OS.

TASK Object

Parameters of TASK object type define the task properties. The syntax of the task object definition is as follows:

```
TASK <name of TASK> {
        PRIORITY = <integer>;
        SCHEDULE = <FULL / NON>;
        AUTOSTART = <TRUE / FALSE>{
          APPMODE = <name of APPMODE>;
        };
        ACTIVATION = <1>;
        STACKSIZE = <integer>;
        RESOURCE = <name of RESOURCE>;
        EVENT = <name of EVENT>;
        ACCESSOR = <SENT / RECEIVED> {
            MESSAGE = <name of MESSAGE>;
            WITHOUTCOPY = <TRUE / FALSE>;
            ACCESSNAME = <string>;
```

```
        };
};
```

The brief description of the task attributes is presented below.

Table A.8 TASK Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
PRIORITY	integer [0..0x7FFFFFFF]	Defines the priority of the task. The lowest priority has value 0
SCHEDULE	FULL, NON	Defines the run-time behavior of the task
AUTOSTART	TRUE, FALSE	Defines whether the task is activated during the system start-up procedure or not
APPMODE	name of APPMODE	Defines an application mode in which the task is auto-started
ACTIVATION	1	Specifies the maximum number of queued activation requests for the task. The OSEKturbo OS does not support multiple activation, so this value is restricted to 1
RESOURCE	name of RESOURCE	Resources accessed by the task. There can be several resource references
EVENT	name of EVENT	Events owned by the task. There can be several event references
ACCESSOR	SENT, RECEIVED	Defines the type of usage for the message
MESSAGE	name of MESSAGE	Specifies the message to be sent or received by the task
WITHOUTCOPY	TRUE, FALSE	Defines whether a local copy of the message is used or not
ACCESSNAME	string	Defines the reference which can be used by the application to access the message data
OSEKturbo Specific Attribute		
STACKSIZE	integer	Defines the size of the extended task's stack in bytes

ISR Object

This object represents an Interrupt Service Routine. Parameters of this object type define ISR properties. The syntax of the ISR object is as follows:

```
ISR <name of ISR> {
        CATEGORY = <1 / 2>;
        PRIORITY = <integer>;
        RESOURCE = <name of RESOURCE>;
        ACCESSOR = <SENT / RECEIVED> {
          MESSAGE = <name of MESSAGE>;
          ACCESSNAME = <string>;
        };
};
```

The following parameters can be defined for the ISR object:

Table A.9 ISR Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
CATEGORY	1, 2	Specifies the category of interrupt service routine
RESOURCE	name of RESOURCE	Specifies the list of resources accessed by the task. The reference can not be defined if *CATEGORY* is 1.There can be several resource references
ACCESSOR	SENT, RECEIVED	Defines the type of usage for the message
MESSAGE	name of MESSAGE	Specifies the message to be sent or received by the ISR
ACCESSNAME	string	Defines the reference which can be used by the application to access the message data
OSEKturbo Specific Attributes		
PRIORITY	[0-16] or [0-48] for Enhanced *InterruptDispatcher*	Specifies the priority of the interrupt service routine

RESOURCE Object

The RESOURCE object is intended for the resource management. The syntax of the resource object is as follows:

```
RESOURCE <name of resource> {
  RESOURCEPROPERTY = <STANDARD / LINKED /  INTERNAL> {
    LINKEDRESOURCE = <name of RESOURCE>
```

```
  };
};
```

The following standard parameters can be defined for the RESOURCE object:

Table A.10 RESOURCE Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
RESOURCEPROPERTY	STANDARD, LINKED, INTERNAL	Specifies a property of the resource. Performance decreases if RESOURCE with RESOURCEPROPERTY = INTERNAL defined
LINKEDRESOURCE	name of RESOURCE	Specifies the resource to which the linking shall be performed

EVENT Object

The EVENT object is intended for the event management. The syntax of the event object is as follows:

```
EVENT <name of EVENT> {
      MASK = <integer / AUTO>;
};
```

The following standard parameters can be defined for the EVENT object:

Table A.11 EVENT Parameters

Object Parameters	Possible Values	Description
Standard Attribute		
MASK	integer, **AUTO**	Represents the event

COUNTER Object

Attributes of this object type define counter properties. The syntax of the counter object is:

```
COUNTER <name of COUNTER> {
      MINCYCLE = <integer>;
```

```
      MAXALLOWEDVALUE = <integer>;
      TICKSPERBASE = <integer>;
};
```

The following standard parameters can be defined for the COUNTER object:

Table A.12 COUNTER Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
MINCYCLE	integer	Specifies the minimum allowed number of counter ticks for a cyclic alarm linked to the counter
MAXALLOWEDVALUE	integer	Defines the maximum allowed counter value
TICKSPERBASE	integer	Specifies the number of ticks required to reach a counter-specific value

ALARM Object

This object presents OS alarms. The syntax of an alarm object is as follows.

```
ALARM <name of ALARM> {
  COUNTER = <name of COUNTER>;
  ACTION = <SETEVENT / ACTIVATETASK / ALARMCALLBACK> {
    TASK = <name of TASK>;
    EVENT = <name of EVENT>;
    ALARMCALLBACKNAME = <string>;
  };
  AUTOSTART = <TRUE / FALSE> {
    ALARMTIME = <integer>;
    CYCLETIME = <integer>;
    APPMODE = <name of APPMODE>;
  };
};
```

The following standard parameters can be defined for the ALARM object:

Table A.13 ALARM Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
COUNTER	name of COUNTER	Specifies the assigned counter
ACTION	ACTIVATETASK, SETEVENT, ALARMCALLBACK	Defines the method of notification used when the alarm expires
TASK	name of TASK	Specifies the task being notified through activation or event setting when the alarm expires
EVENT	name of EVENT	Specifies the event mask to be set when the alarm expires. It shall be defined if ACTION is SETEVENT only
ALARMCALLBACKNAME	string	Specifies the name of the callback routine called when the alarm expires
AUTOSTART	TRUE, FALSE	Defines whether an alarm is started automatically at system start-up depending on the application mode
ALARMTIME	integer	Defines the time when the alarm shall expire first
CYCLETIME	integer	Defines the cycle time of a cyclic alarm
APPMODE	name of APPMODE	Defines an application mode in which the alarm will be started automatically at system start-up

MESSAGE Object

Parameters of this object type define the message properties. The syntax of the message object definition is presented below. Note that only one *ACTION* attribute should be defined for the MESSAGE object.

```
MESSAGE <name of MESSAGE> {
    TYPE = <QUEUED / UNQUEUED>;
      QUEUEDEPTH = <integer>;
    CDATATYPE = <string>;
    ACTION = <ACTIVATETASK / SETEVENT / CALLBACK / FLAG / NONE> {
        TASK = <name of TASK>;
        EVENT = <name of EVENT>;
        CALLBACKNAME = <string>;
        FLAGNAME = <string>;
```

```
    };
};
```

The following standard parameters can be defined for the MESSAGE object:

Table A.14 MESSAGE Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
TYPE	QUEUED, UNQUEUED	Specifies the message type
QUEUEDEPTH	integer	Specified if the message has a queue
CDATATYPE	string	Defines the data type of a message item
ACTION	ACTIVATETASK, SETEVENT, CALLBACK, FLAG, NONE	Defines the type of task notification used when the message has arrived
TASK	name of TASK	Specifies the task which shall be notified when the message has arrived. It shall be defined if *ACTION* is *ACTIVATETASK* or *SETEVENT* only
EVENT	name of EVENT	Specifies the event to be set when the message has arrived. It shall be defined if *ACTION* is *SETEVENT* only
CALLBACKNAME	string	Defines the name of a function to call as an action when the message has been sent. It shall be defined if *ACTION* is *CALLBACK* only
FLAGNAME	string	Defines the name of the flag that is set when the message is sent. It shall be defined if *ACTION* is *FLAG* only

APPMODE Object

The APPMODE object is intended for the application mode management. This object has no standard parameters.

COM Object

The COM object represents the OSEK communication subsystem properties on CPU. Only one COM object must be defined on the local CPU. The syntax scheme of a COM object is as follows:

```
COM <name of COM> {
    USEMESSAGERESOURCE = <TRUE / FALSE>;
    USEMESSAGESTATUS = <TRUE / FALSE>;
};
```

The object has the following standard attributes:

Table A.15 COM Parameters

Object Parameters	Possible Values	Description
Standard Attributes		
USEMESSAGERESOURCE	TRUE, FALSE	Specifies if the message resource mechanism is used
USEMESSAGESTATUS	TRUE, FALSE	Specifies if the message status is available

NM Object

The NM object represents the local parameters of the network management subsystem on CPU. This object has no standard parameters.